JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Development of a Modbus RTU Configuration Tool for RS-485 Serial Communication Protocol

[1]Koti Abhinaya, [2]Chokkapu Ravindra Naidu.
[1]Student, [2]Student.
[1]dept of Electronics and Communication Engineering,
[1]R.V.R & J.C College of Engineering, Guntur, India.

Abstract: This project presents a Modbus setting Tool that uses the RS-485 serial communication protocol to simplify the setting of industrial devices. By utilising the widely-accepted Modbus protocol, the programme provides an easy-to-use interface designed for the management of many windows, each of which corresponds to a separate network device ID. It guarantees dependable communication between the master device and different slaves by using the efficiency and resilience of RS-485, enabling real-time interaction for configuration, monitoring, and diagnostics. Important features include real-time response handling for timely feedback and a modular multi-window interface that allows for the simultaneous operation of multiple devices. The tool's user-friendly design makes it easier to use, and its integrated error detection methods help with fault isolation and debugging. All things considered; the Modbus settings Tool is a useful tool for industrial automation that optimises device settings.

***Index Terms* – MODBUS- RTU, RS-485, Python, Data Converters, PYQT5, GUI, Serial Port Communication.**

I. Introduction

A. Modbus-RTU.

One of the most often used versions of the Modbus communication protocol is the Modbus RTU (Remote Terminal Unit). It is extensively used in industrial automation systems to facilitate communication between actuators, sensors, PLCs, and other industrial devices. "Remote Terminal Unit," or RTU, is an acronym that denotes a device's ability to serve as a communication bridge between a master device (such a supervisory control and data acquisition system, or SCADA) and slave or remote terminal units. When using the Modbus RTU protocol, serial communication—typically over RS-485 or RS-232 interfaces—is used to transfer data in binary format. Because of its differential signalling, which provides higher noise protection and longer cable lengths than RS-232, RS-485 is especially preferred. Modbus RTU is hence appropriate for industrial settings where robust communication is essential, and devices may be distributed across large distances.

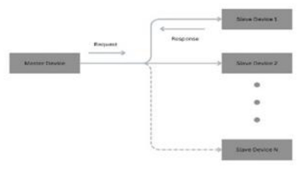


Fig1: Master-Slave communication

A single master device initiates communication by issuing requests to one or more slave devices in the master-slave architecture used by Modbus RTU. Usually, these requests entail reading or writing information to/from particular memory registers on the slave devices. The addressed slave device receives a request, processes it, and replies to the master appropriately. The effectiveness and simplicity of Modbus RTU are two of its main characteristics. The device address, function code, data field, and error checking field (such as CRC or checksum) make up the compact frame structure used by the protocol. since of its lightweight nature, Modbus RTU is ideal for real-time control and monitoring since it reduces overhead and facilitates quick communication between devices.

B.RS-485 Serial Communication.

In both industrial and commercial settings, RS-485 is a popular serial communication standard that is renowned for its durability, dependability, and adaptability. The RS-232, RS-422, and RS-423 standards are among the family of serial communication standards to which it belongs. Because of its flexibility for multi-point networks, extended cable lengths, and noise tolerance, RS-485 is very well-liked in industrial settings.

Table1:RS-485 characteristics table

Characteristics	RS-485
Max distance	1200m
Max Speed	100kbps
Mode of Operation	Full duplex/Half duplex
Max number of drivers	32
Max number of receiver's	32
Network topology	Multi Point

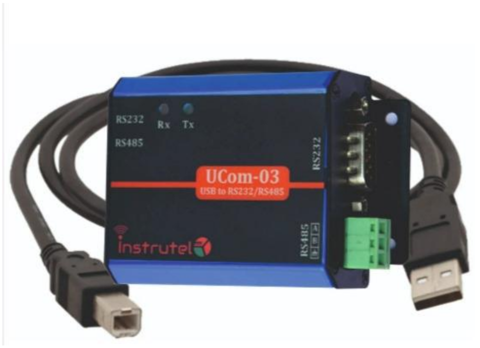


Fig2:RS-485 Data converter

All things considered, RS-485 serial communication provides an effective and dependable way to link numerous devices across great distances in commercial and industrial contexts. It is an essential component of contemporary communication systems due to its high data rates, support for multi-point networks, and resistance to noise.

II. Design Phase.

There will be three steps in the development of the modbus-rtu setup tool using the RS-485 serial communication protocol.
A. Using Python's PyQt5 to design the graphical interface.
B. Modbus Parameter configuration.
C. The format of data frame structure.
By following these steps, you will have created a full Modbus RTU configuration tool that enables users to create data frames and interactively set up communication parameters for data exchange with Modbus RTU devices over RS-485 serial connection.

A. Using Python's PyQt5 to design the graphical interface.

- Any configuration tool must have a GUI (Graphical User Interface) developed since it gives users an easy-to-use interface for interacting with the programme.
- A robust framework for building Python cross-platform graphical user interfaces is called PyQt5. It offers tools for constructing several GUI components, including input fields, buttons, and labels.
- This step involves designing the user interface for your programme, including all the widgets and controls needed to configure the Modbus RTU parameters.

B. Modbus Parameter configuration.

In a Windows environment, software programmes can loop over numerical values beginning with COM1 and try to access each port to see if it is available for use. A port is deemed available for communication with Modbus devices once it has been successfully opened.
A few important settings need to be taken into account when defining Modbus parameters:
Baud Rate: The number of symbols transmitted per second is indicated by this option, which controls the pace of communication the RS-485 network. Baud rates of 9600, 19200, and 38400 bits per second are common. (fig3)
Parity: Parity adds an extra bit (None, Even, or Odd) to each character that is transmitted in order to offer rudimentary error detection. During transmission, it aids in maintaining data integrity. (fig4)
Data Bits: This option indicates how many bits are utilised to represent each.
Stop Bits: These bits let the transmitter and receiver synchronise by indicating the end of a character. In Modbus communication, one or two stop bits are typically utilised.

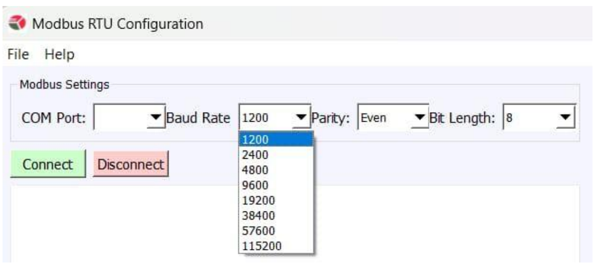

Modbus RTU Configuration
COM Port: Baud Rate 1200 Parity: Even Bit Length: 8

fig3: Baud Rate parameters setting

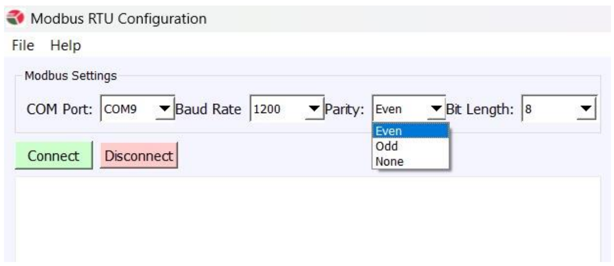

Modbus RTU Configuration
COM Port: COM9 Baud Rate 1200 Parity: Even Bit Length: 8

fig4: Parity parameters settings

C. The format of data frame structure.

Each data frame in Modbus RTU communication adheres to a predetermined format to provide dependable data transfer between devices. An outline of the elements that make up a Modbus RTU message is as follows:
Starting Address: The memory address in a slave device that the master device reads or writes data to is referred to as the starting address in Modbus RTU. Below Table 3 shows the starting address range for its Register type.
Function Code: The type of Modbus function being requested or the action that the slave device is to carry out is specified in this field. Function codes can be used to write multiple registers, write single coils, read holding registers, and read coils, among other things.
Data: The additional information needed for the given function code is contained in the data field. It could contain register addresses, values to write, or the total number of registers to read or write for read/write operations.

CRC (Cyclic Redundancy Check): An error-checking technique called the CRC field is used to guarantee the integrity of the data while it is being transmitted. Prior to transmission, it is attached to the end of the message and calculated based on its contents. The CRC is computed after receiving a message; if it differs from the received CRC, a transmission error has occurred.

Table2: Modbus RTU Master data frame

FIELD	SIZE(BYTES)
Device ID	1
Starting Address	2
Function Code	1
Num of Registers	2
CRC	2

Table3: Register Table

REGISTER TYPE	REGISTER ADDRESS	FUNCTION CODE
Coil Registers	1-9999	01
Discrete Inputs	10001-19999	02
Input Registers	30001-39999	04
Holding Registers	40001-49999	03

In order to create a Modbus RTU configuration tool, your software must be able to produce valid Modbus RTU messages from user inputs such as data fields, address, and function code. In order to guarantee data integrity, it should also be able to parse received messages in order to extract pertinent data and carry out error checking using CRC validation. To accomplish this, you must incorporate the Modbus RTU protocol specifications into the communication routines of your programme.

III. Data communication between master and slave devices.

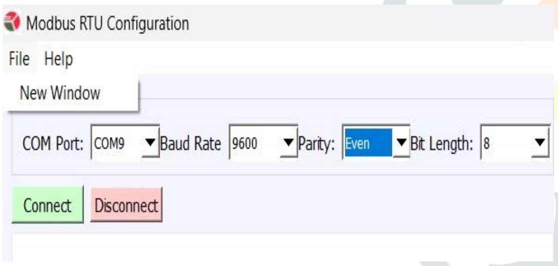
These steps delineate the step-by-step procedure of communication in a Modbus RTU system between the master and slave devices, encompassing request initiation, response handling, and data presentation through the user interface of the Modbus setup tool.

A. Sending Request from Master Device.

B. Slave response Section

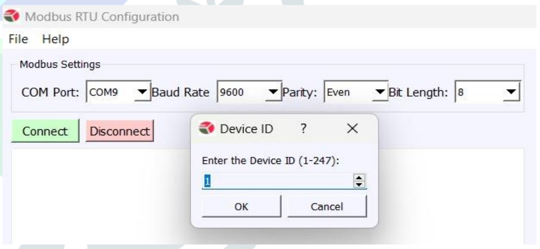
C. Handling Multiple Device ID windows and Displaying Data.

In a Modbus RTU network, these steps describe the sequential communication process between master and slave devices as well as the user interface concerns for the software application on the master device. In the context of the Modbus network, each step is essential to the smooth operation of user interaction and data exchange.



COM Port: COM9 Baud Rate 9600 Parity: Even Bit Length: 8

Fig5: Opening New device Window



COM Port: COM9 Baud Rate 9600 Parity: Even Bit Length: 8

Enter the Device ID (1-247): 1

Fig6: Setting Device ID

A. Sending Request from Master Device:

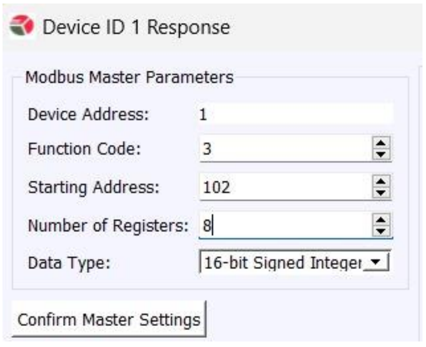
The master device sends a request to the slave device at the beginning of the data transmission process between the master and slave devices.

I) Master Setting Configuration:

- The user configures the master device settings prior to issuing a request. Typically, this entails giving details about the baud rate for communication, the parity setting for error checking, the COM port that the master device is linked to, etc.
- By using these configurations, the RS-485 network's master and slave devices are guaranteed to communicate and synchronise properly.

II) Confirming Master Settings:

- Through the application interface, the user verifies the master settings after they have been configured. Sending the request packet to the slave device is signalled by this action to the application.
- The application can move forward with the transmission procedure after the confirmation phase verifies that the user is prepared to start a conversation.



Device ID 1 Response

Modbus Master Parameters

Device Address: 1
Function Code: 3
Starting Address: 102
Number of Registers: 8
Data Type: 16-bit Signed Integer

Fig7: Master settings

III) Sending Request Packet:

- The application creates a Modbus RTU packet with the required data for the request after verifying the master parameters.
- Usually, this packet contains:
- The RS-485 network address of the slave device.
- The function code (e.g., read coil, read holding register, etc.) designating the kind of request to be executed.
- Depending on the function code, more information may be required, such as register addresses, or the quantity of registers to read.
- A value for the Cyclic Redundancy Check (CRC) to detect errors.
- After the packet is assembled, it is stored in Queue, then pop packets sent, with the defined master settings, to the targeted slave device via the RS-485 network.

B. Slave response Section:

The Modbus communication mechanism relies heavily on the slave response step to ensure dependable data transfer between the slave and master devices.

I) Assessing Modbus Parameters and CRC Verification:

- The addressed slave device first assesses the Modbus parameters in the request packet after receiving it from the master device. These parameters contain data like the function code, the slave device's address, and any other relevant information. When a packet is received, the slave device uses a CRC (Cyclic Redundancy Check) to ensure that the data transmission was completed successfully. The CRC value contained in the request packet and the value computed by the slave should line up.

II) Taking the Request and Performing the Function:

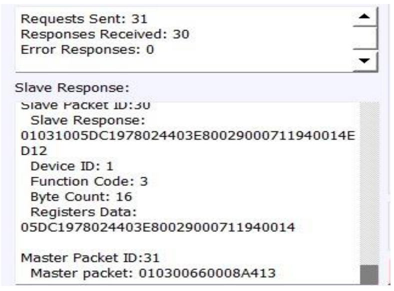
- The slave device approves the request from the master device if the Modbus parameters are acceptable and the CRC check is successful. After that, the function code in the request packet on the slave device indicates which Modbus function to execute. Depending on the function code, this could include actions like writing to or reading from registers or conducting control operations.

III)Framing the response Packet:

- The slave device creates a response packet with the desired data or acknowledgment information after processing the request and carrying out the designated function. Usually, the response packet consists of: The slave device's address. The function code designating the kind of answer (such as an error response, read response, among others). The data payload, which could include error codes, status details, or register values. a CRC value for error detection that is computed across the whole answer packet.

IV) Sending the Master Device the Response:

- The slave device uses the RS-485 network to communicate the response packet back to the master device after it has been created. After receiving the response packet, the master device processes the data or acknowledgment information that is included in it.



```
Requests Sent: 31
Responses Received: 30
Error Responses: 0

Slave Response:
Slave Packet ID:30
  Slave Response:
01031005DC1978024403E80029000711940014E
D12
  Device ID: 1
  Function Code: 3
  Byte Count: 16
  Registers Data:
05DC1978024403E80029000711940014

Master Packet ID:31
  Master packet: 010300660008A413
```

Fig8: Slave response

The acknowledgement of the request, the execution of the designated Modbus function, the creation of the response packet, and the transmission of the response back to the master device are all included in the slave response stage. In a Modbus RTU network, this step is essential for maintaining dependable and seamless communication between the slave and master devices.

C. Handling Multiple Device ID windows and Displaying Data.

Managing several windows and presenting information is essential to giving people a thorough understanding of the Modbus communication process.

I) Obtaining and Interpreting the Response Packet:

- The master device extracts the data from the response packet after receiving it from the slave device. In order to interpret the data payload, which could contain error codes, status information, or register values, this requires parsing the response packet. After parsing, the data is processed before being shown to the user.

II) Displaying Data in User-Friendly Format:

- The data that has been received is presented in an easy-to-use style together with pertinent details like values and data addresses.
- The way the data is presented in an application may vary, but it may include tables.
- Accessing and interpreting the provided data should be a breeze for users thanks to a straightforward and navigable user interface.

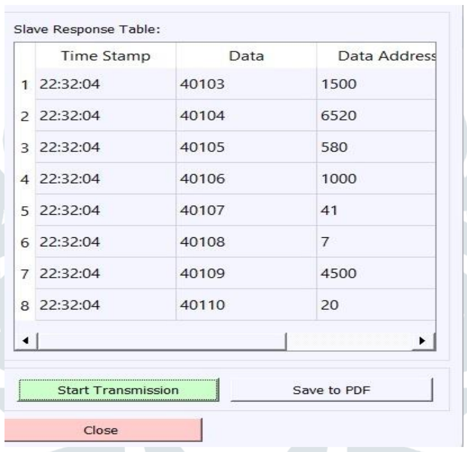
III) Managing Data by Using Several Windows:

- The application may use several windows or tabs to efficiently handle and display various kinds of data.
- Aspects of Modbus communication like as coil values, register values, or diagnostic data might be the emphasis of each window or tab. This kind of data organisation improves usability and enables users to concentrate on the information most pertinent to their tasks.

IV) Organised Perspective with Slave Response Table:

- Users can see the received data in a clear and organised manner with the help of a structured display, such a table. Columns for data addresses, values, and any other information pertinent to the Modbus communication process are usually present in the slave response table. With the help of this organised view, users may efficiently monitor the Modbus communication process by reducing the complexity of data interpretation and analysis.

.

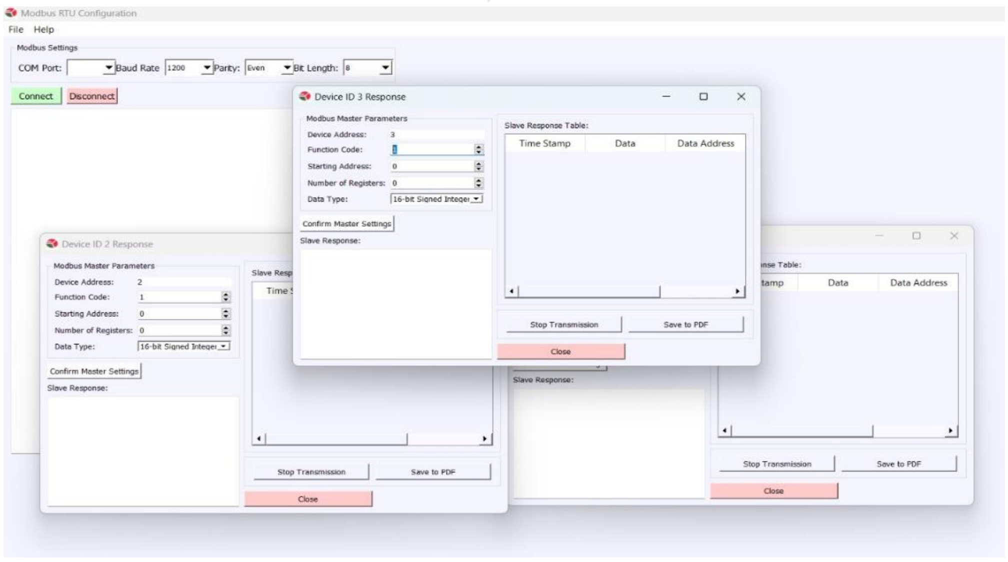

Slave Response Table:

	Time Stamp	Data	Data Address
1	22:32:04	40103	1500
2	22:32:04	40104	6520
3	22:32:04	40105	580
4	22:32:04	40106	1000
5	22:32:04	40107	41
6	22:32:04	40108	7
7	22:32:04	40109	4500
8	22:32:04	40110	20

Fig9: Slave Response Table

V) Facilitating the Process of Monitoring, Analysis, and Control:

- This step's main goal is to efficiently convey the data to the user so that the Modbus communication process may be monitored, analysed, and controlled. Users are able to track data in real time, examine past trends, and act appropriately in response to the information presented.


Modbus RTU Configuration

Modbus Settings: COM Port: ; Baud Rate: 1200; Parity: Even; Bit Length: 8

Device ID 3 Response

Modbus Master Parameters
Device Address: 3
Function Code: 1
Starting Address: 0
Number of Registers: 0
Data Type: 16-bit Signed Integer

Slave Response Table:

Time Stamp	Data	Data Address

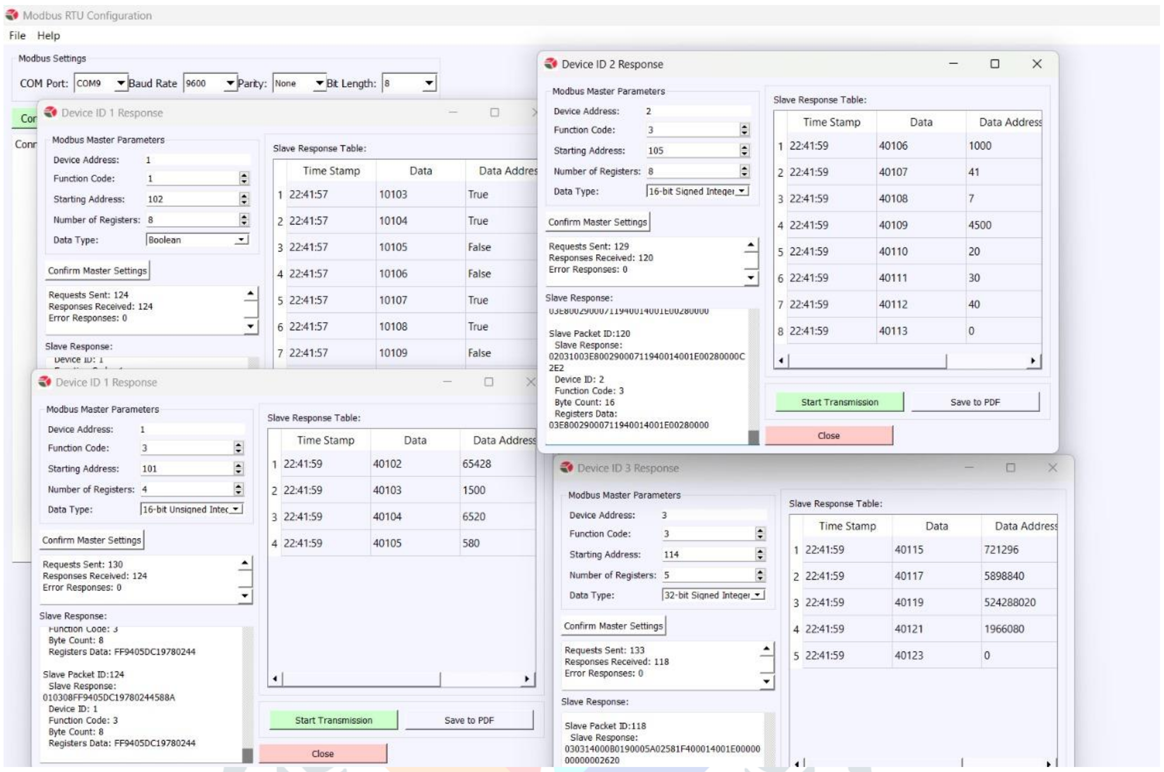
Device ID 2 Response

Modbus Master Parameters
Device Address: 2
Function Code: 1
Starting Address: 0
Number of Registers: 0
Data Type: 16-bit Signed Integer

Fig10: Output for handling multiple windows

IV)Testing Multiple Device windows:

- Through the use of numerous windows, each corresponding to a distinct data address or device, this information may be presented to users so they can effectively monitor and interact with various Modbus network elements at the same time.
- With this configuration, the tool's capacity to manage numerous data streams efficiently and offer real-time insights into the industrial system's performance can be thoroughly tested and validated. Shown in fig11.



COM Port: COM9 Baud Rate 9600 Parity: None Bit Length: 8

Device ID 1 Response

Modbus Master Parameters

Device Address: 1
Function Code: 1
Starting Address: 102
Number of Registers: 8
Data Type: Boolean

Confirm Master Settings

Requests Sent: 124
Responses Received: 124
Error Responses: 0

Slave Response:

Device ID: 1

Slave Response Table:

	Time Stamp	Data	Data Address
1	22:41:57	10103	True
2	22:41:57	10104	True
3	22:41:57	10105	False
4	22:41:57	10106	False
5	22:41:57	10107	True
6	22:41:57	10108	True
7	22:41:57	10109	False

Device ID 2 Response

Modbus Master Parameters

Device Address: 2
Function Code: 3
Starting Address: 105
Number of Registers: 8
Data Type: 16-bit Signed Integer

Confirm Master Settings

Requests Sent: 129
Responses Received: 120
Error Responses: 0

Slave Response:

```
03E800290007119400140 01E00280000

Slave Packet ID:120
  Slave Response:
02031003E80029000711940014001E00280000C
2E2
  Device ID: 2
  Function Code: 3
  Byte Count: 16
  Registers Data:
03E80029000711940014001E00280000
```

Slave Response Table:

	Time Stamp	Data	Data Address
1	22:41:59	40106	1000
2	22:41:59	40107	41
3	22:41:59	40108	7
4	22:41:59	40109	4500
5	22:41:59	40110	20
6	22:41:59	40111	30
7	22:41:59	40112	40
8	22:41:59	40113	0

Device ID 1 Response

Modbus Master Parameters

Device Address: 1
Function Code: 3
Starting Address: 101
Number of Registers: 4
Data Type: 16-bit Unsigned Integer

Confirm Master Settings

Requests Sent: 130
Responses Received: 124
Error Responses: 0

Slave Response:

```
  Function Code: 3
  Byte Count: 8
  Registers Data: FF9405DC19780244

Slave Packet ID:124
  Slave Response:
010308FF9405DC19780244588A
  Device ID: 1
  Function Code: 3
  Byte Count: 8
  Registers Data: FF9405DC19780244
```

Slave Response Table:

	Time Stamp	Data	Data Address
1	22:41:59	40102	65428
2	22:41:59	40103	1500
3	22:41:59	40104	6520
4	22:41:59	40105	580

Device ID 3 Response

Modbus Master Parameters

Device Address: 3
Function Code: 3
Starting Address: 114
Number of Registers: 5
Data Type: 32-bit Signed Integer

Confirm Master Settings

Requests Sent: 133
Responses Received: 118
Error Responses: 0

Slave Response:

```
Slave Packet ID:118
  Slave Response:
030314000B0190005A02581F400014001E00000
00000002620
```

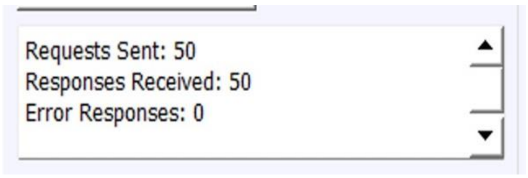
Slave Response Table:

	Time Stamp	Data	Data Address
1	22:41:59	40115	721296
2	22:41:59	40117	5898840
3	22:41:59	40119	524288020
4	22:41:59	40121	1966080
5	22:41:59	40123	0

Fig11: Handling data in multiple windows

A potent tool in software development, threading allows for better scalability, concurrent job execution, and the design of user interfaces that are responsive. The following is how threading supports each of these elements:

V) Error Handling and Debugging:
Adding methods to capture and handle failures gracefully, along with debugging information to help with troubleshooting, will allow you to implement error handling and debugging into the offered code sample.
A. Error Handling: The process of foreseeing, identifying, and resolving problems or unusual circumstances that can arise during programme execution is known as error handling. Error handling in this context seeks to guarantee that the programme keeps working correctly and gracefully navigates any unforeseen circumstances that may occur.
I)Try-Except Block:
In Python, an attempt-except block is a structure used to capture and manage exceptions that arise inside a designated code block. The code that could throw an exception is located in the try block. The interpreter searches for an except block that can handle a specific exception if it arises inside the try block. The code within the except block is performed to handle the exception if a matching exception handler is found. Depending on the circumstances, the exception either propagates to the outer scope or results in the programme terminating if no matching exception handler is identified.
B) Debugging:
It is the process of locating and fixing flaws or errors in a Slave Device, it entails methodically identifying the source of unexpected behaviour and fixing it.



Requests Sent: 50
Responses Received: 50
Error Responses: 0

Fig12: Error Response

To summarise, error handling and debugging are crucial components of software development that guarantee programme dependability and maintainability. Try-except blocks are utilised in error handling for debugging, which helps technicians to find, analyse errors during programme execution.

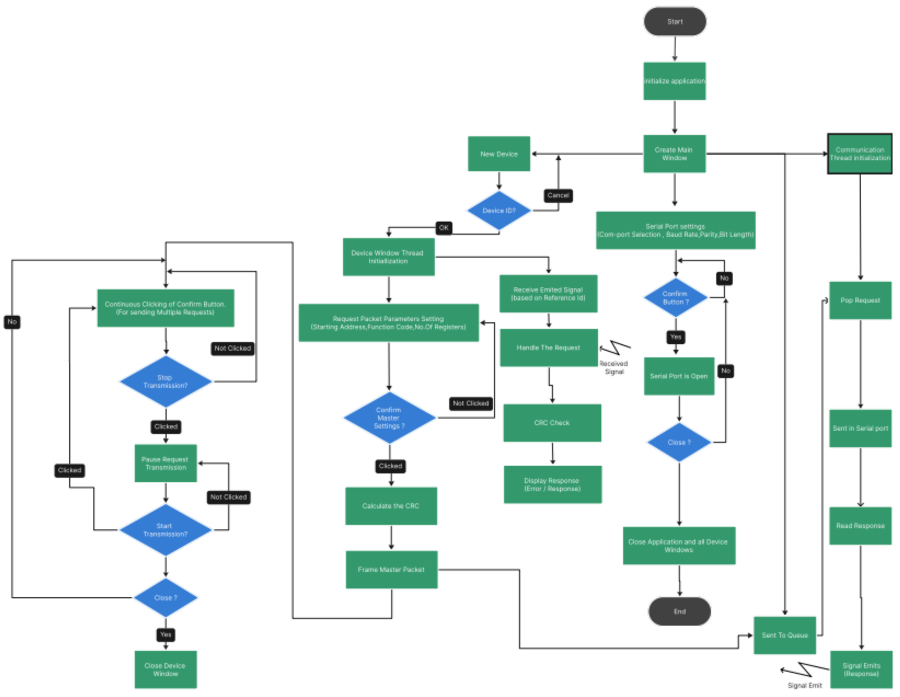

Start
initialize application
Create Main Window
New Device
Device ID?
Cancel
OK
Communication Thread initialization
Serial Port settings (Com-port Selection , Baud Rate,Parity,Bit Length)
Confirm Button ?
No
Yes
Serial Port is Open
No
Close ?
Close Application and all Device Windows
End
Device Window Thread Initialization
Receive Emited Signal (based on Reference Id)
Handle The Request
Received Signal
CRC Check
Display Response (Error / Response)
Request Packet Parameters Setting (Starting Address,Function Code,No.Of Registers)
Confirm Master Settings ?
Not Clicked
Clicked
Calculate the CRC
Frame Master Packet
Continuous Clicking of Confirm Button. (For sending Multiple Requests)
No
Not Clicked
Stop Transmission?
Clicked
Pause Request Transmission
Clicked
Not Clicked
Start Transmission?
Close ?
Yes
Close Device Window
Pop Request
Sent in Serial port
Read Response
Sent To Queue
Signal Emits (Response)
Signal Emit

Fig12: Complete Architecture and Flow chart for the Modbus-RTU tool.

VI)References:

1. Modbus Organization, ‖ Modbus Technical Resources. [Online]. Available: http://modbus.org/tech.php. [Accessed:20-Jan-2024]
2. Modbus RTU Protocol Overview, ‖ Real Time Automation, Inc., 25-Nov-2019. [Online]. Available: https://www.rtaautomation.com/technologies/modbus-rtu/ [Accessed:24-Jan-2024]
3. Kasun Herath, Hasanika Samarasinghe, Hapuarachchige Don Nelaka Shayamal Priyankara “Development of a Data Acquisition and Monitoring System Based on MODBUS-RTU Communication Protocol “Conference Paper · June 2020. [Accessed:28-Jan-2024]
4. Distantcity, ―Simple Modbus Protocol in C# / .NET 2.0,‖ Code Project, 18-Oct-2007. [Online]. Available: https://www.codeproject.com/Articles/20929/Simple-Modbus-Protocolin-C-NET. [Accessed: 12-Feb-2024].
5. Modbus Organization, Inc. Modbus Application Protocol Specification V1.1b3, ‖ http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf[Accessed: 20-Feb-2024].